列表(List)

列表是计算机中一种常见的数据结构，日常生活中的购物清单，待办事项等都可以成为列表，它是一组有序的数据，每个列表中的数据项称为**元素**。在javascript中，列表中的元素可以是任意数据类型。列表中可以保存多少元素并没有限定（在实际使用时会受到程序内存的限制）。

列表中会有一些常见属性或方法，比如列表中的元素个数，列表当前的位置，向列表末尾增加一个元素，从列表中删除一个元素，清空列表等一系列操作。

接下来，我们抽象出一个列表的数据类型定义，并用JS中的数组去实现它。

类型	描述
listSize(属性)	列表元素个数
pos(属性)	列表的当前位置
length(属性)	清空列表中的所有元素
clear(方法)	清空列表所有元素
toString(方法)	返回列表的字符串形式
getElement(方法)	返回当前位置的元素
insert(方法)	在现有的元素后插入新元素
append(方法)	在列表的末尾添加新元素
remove(方法)	从列表中删除元素
front(方法)	将列表的当前位置移动到第一个元素
end(方法)	将列表的当前位置移动到最后一个元素
prev(方法)	将当前位置前移一位
next(方法)	将当前位置后移一位
currPos(方法)	返回列表的当前位置
moveTo(方法)	将当前位置移动到指定位置
contains(方法)	判断给定值是否在列表中

列表的数据类型定义

列表类

```
/*定义List类*/
 function List () {
    this.listSize = 0;    //初始化元素个数为0
    this.pos = 0;         //初始化位置为0
    this.dataStore = []; //初始化空数组来保存列表元素
    this.clear = clear;
    this.find = find;     //寻找元素
    this.toString = toString; //显示列表中的元素
    this.insert = insert;
    this.append = append;
    this.remove = remove;
    this.front = front;
    this.end = end;
    this.prev = prev;
    this.next = next;
    this.length = length;  //列表中的元素总数
    this.currPos = currPos;
    this.moveTo = moveTo;
    this.getElement = getElement;
    this.contains = contains; //判断给定值是否在列表中
}
```

接着我们来实现这些方法。
首先得可以添加元素，所以我们先来编写 append 方法
append：向列表中添加一个元素

```
//该方法给列表的最后一个位置添加一个新的元素，待插入成功后，更新列表中的元素个数

function append ( element ) {
    this.dataStore[this.listSize++] = element;
}
```

这样我们就可以往列表里面添加元素了，接着我们来看查找 find 方法

find：查找列表中的某一个元素

```
//该方法通过循环查找给定元素是否在列表中，如果存在返回元素的位置，否则返回 -1

function find(element){
    for( var i = 0 ; i < this.dataStore.length ; i ++ ){
            if( this.dataStore[i] == element ){
                return i;
            }
        }
    return -1;
}
```

可以插入元素，那总得可以删除了，我们利用 remove 方法删除一个元素

remove：删除列表中的某一元素

```
//该方法通过使用find()方法返回元素的位置对 dataStore 数组进行截取，如果删除成功，返回 true ，并将 listSize 的值减1，更新列表长度，否则返回 false

function remove ( element ) {
    var foundAt = this.find(element);
    if( foundAt > -1 ){
        this.dataStore.splice( foundAt , 1 );
        --this.listSize;
        return true;
    }
    return false;
}
```

我想知道我的列表还有多少个元素的时候，就得构建 length 方法，

length：返回列表中总的元素个数

```
//该方法直接将 listSize 返回即可

function length(){
    return this.listSize;
}
```

如果能显示我的列表元素就好了，这个可以有，我们来看看 toString 方法，

toString：显示列表的元素

```
//该方法直接返回了列表数组，显示出当前列表内容

function toString(){
    return this.dataStore;
}
```

现在我们的列表已经有了基本的一些功能，不妨下试试

```
//构造列表对象
var fruits = new List();
//添加三个元素
fruits.append('Apple');
fruits.append('Grape');
fruits.append('Banana');

//打印列表
console.log( fruits.toString() )      // ["Apple", "Grape", "Banana"]
//查看列表长度
console.log( fruits.length() )        // 3
//查找 Banana 的位置
console.log( fruits.find('Banana') )  // 2
//删除 Grape
fruits.remove('Grape');
console.log( fruits.toString() )      // ["Apple", "Banana"]
```

如果我们删除了 Grape 元素， 我们还想将它放回到原来的位置， 这时候我们就需要调用 insert 方法

insert：向列表某个位置添加一个元素

```
//该方法需要传入两个参数，第一个参数表示待插入的元素，第二个参数表示待插入元素的前一个元素，用于确定插入元素的位置，并调用 splice 方法更改列表数组，插入成功后更新 listSize 并返回 true ，否则返回 false：
function insert( element , after ){
    var insertPos = this.find( after );
    if( insertPos > -1 ){
        this.dataStore.splice( insertPos + 1 , 0 , element );
        this.listSize ++;
        return true;
    }
    return false;
}
```

现在，我们把 Grape 放到原来的位置上去;

```
fruits.insert( 'Grape' , 'Apple' );
console.log( fruits.toString() )        // ["Apple", "Grape", "Banana"]
```

ok，现在已经能够把 Grape 插入到原来的位置了。

如果我吃完了所有水果，我现在想要清空列表，我们就需要一个 clear 方法;

clear：清空列表

```
//该方法使用 delete 操作符删除 dataStore 数组 ， 接着创建新的数组，并将其 listSize 和 pos 初始化设为 1 。

function clear(){
    delete this.dataStore;
    this.dataStore = [];
    this.listSize = this.pos = 0;
}
```

我们不妨试一试。

```
fruits.clear();
console.log( fruits.toString() );      // []
```

成功了！
上面我们看到了列表中有个 pos 属性，表示当前列表所在的位置，我们接下来就围绕它做点事情，让用户可以自由操作列表
front：将列表的位置移到第一个位置上

```
//该方法只要将 pos 置为 0 即可

function front(){
    this.pos = 0 ;
}
```

end：将列表的位置移到最后一个位置上

```
//同上，该方法只要将 pos 置为列表长度减 1 即可，因为数组下标从 0 开始嘛

function end(){
    this.pos = this.listSize - 1;
}
```

prev：将当前位置前移一位

```
//只要将 pos 的位置减 1 即可，但要主要不能越界

function prev(){
```

```
    if( this.pos > 0 ){
        this.pos --;
    }else{
        console.log('您当前已在首位');
    }
}
```

next：将当前位置后移一位

```
//同理，只要将 pos 的位置加 1 即可，但要主要不能越界

function next(){
    if( this.pos < this.listSize - 1 ){
        ++this.pos;
    }else{
        console.log('您当前已在末尾');
    }
}
```

moveTo：将当前位置移动到指定位置

```
//直接改变 pos 的值即可，注意输入的合法性

function moveTo( position ){
    if( position < 0 || position > (this.listSize - 1) ){
        console.log('请输入正确的位置');
    }else{
        this.pos = position;
    }
}
```

我既然可以随意改变我的列表位置，那我总得知道我当前位置在哪里啊，因此我们需要 currPos 和 getElement 两个方法分别返回当前的位置和元素;

currPos：返回列表的当前位置

```
//只需将 pos 直接返回即可

function currPos(){
    return this.pos;
}
```

getElement：返回当前位置的元素

```
//直接取对应的元素就好

function getElement(){
    return this.dataStore[this.pos];
}
```

接着，我们测试一下，首先将列表恢复到清空前的样子，然后我们在多添加几个元素进去。

```
//再添加几个
fruits.append('Pear');
fruits.append('Orange');
fruits.append('Strawberry');
console.log( fruits.toString() );    // ["Apple", "Grape", "Banana", "Pear", "Orange", "Strawberry"]

//我们先看当前的位置和元素
console.log( fruits.currPos() );     // 0
console.log( fruits.getElement() );  // Apple

//我们尝试改变一下
fruits.moveTo( 2 );
fruits.next();
console.log( fruits.currPos() );     // 3
console.log( fruits.getElement() );  // Pear
```

```
fruits.end();
fruits.prev();
console.log( fruits.currPos() );     // 4
console.log( fruits.getElement() );  // Orange
```

至此，我们已经基本完成了列表的所有功能，还差最后一个，判断给定元素是否在列表内，我们这里采用 contains 方法

contains：判断给定值是否在列表中

```
//我们直接利用利用 indexOf() 或者采用跟为高级的 ES6 中的 includes 方法来判断

//ES5
function contains( element ){
    if( this.dataStore.indexOf( element ) > -1 ) return true;
    else return false;
}

//ES6

function contains( element ){
    return this.dataStore.includes( element );
}
```

写完测试一下，

```
console.log(fruits.contains('Banana'));         // true
console.log(fruits.contains('Watermelon'));     // false
```